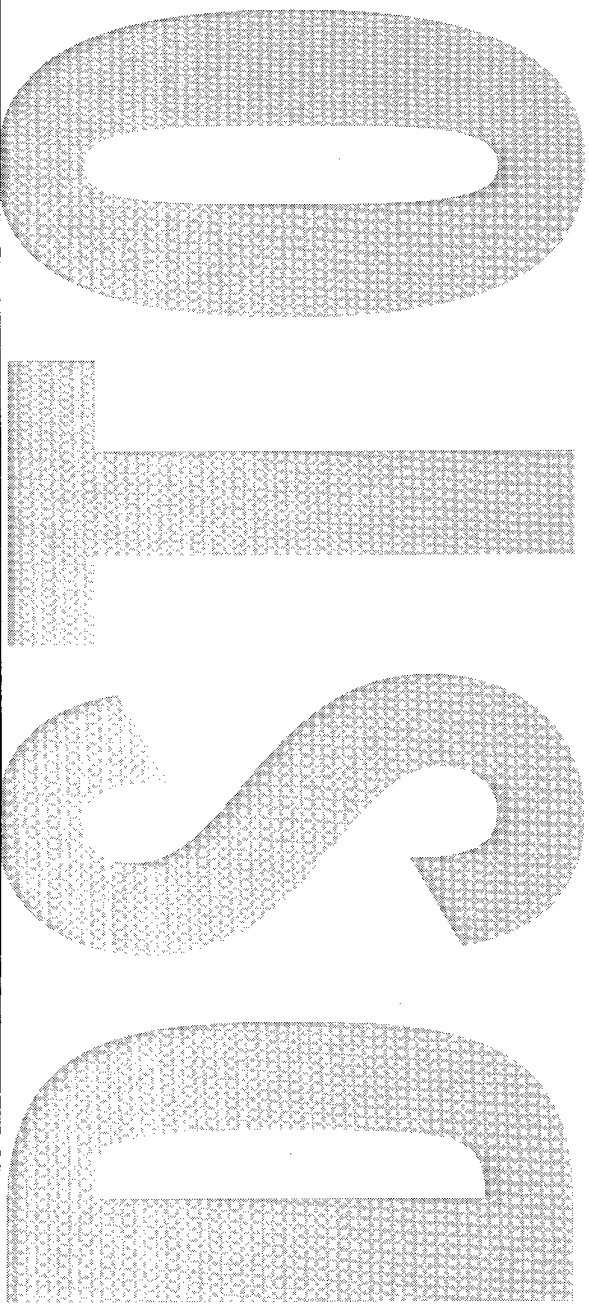
DEPARTMENT OF DEFENCE
DEFENCE SCIENCE & TECHNOLOGY ORGANISATION
DSTO



A Preliminary Study of Open Signalling for ATM Networks

T.A. Au

DSTO-TR-0800

A Preliminary Study of Open Signalling for ATM Networks

T. A. Au

Communications Division
Electronics and Surveillance Research Laboratory

DSTO-TR-0800

ABSTRACT

Based on a clear separation between switching hardware and control software, the concept of open signalling creates an open programmable networking environment. Network entities can be realised as high level objects with well-defined software interfaces, facilitating the creation of multiple mechanisms for connection management. Applying open signalling in defence networks can enhance the flexibility in supporting network services, allowing dynamic control of quality of service for maximum military value. This report discusses the design criteria and the associated performance issues of a connection management system for ATM networks. However, significant binding overheads are generated in remote operation invocations, adding a level of complexity to network control. This complexity is expected to diminish as middleware technology matures, thereby improving the overall performance in connection management.

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

Published by

DSTO Electronics and Surveillance Research Laboratory
PO Box 1500
Salisbury South Australia 5108 Australia

Telephone: (08) 8259 5555
Fax: (08) 8259 6567

AR-010-882
March 1999

APPROVED FOR PUBLIC RELEASE

A Preliminary Study of Open Signalling for ATM Networks

Executive Summary

The key intelligence of an ATM network lies in its signalling system, providing the ability to establish, maintain, and release connections between users and network nodes. This signalling infrastructure not only dominates the ability to introduce new network services, its effectiveness also determines the total performance of a network. The Common Object Request Broker Architecture (CORBA) has recently been recognised as an important enabling technology for the development of distributed applications. The flexibility of CORBA stems from its support for interoperability across heterogeneous hardware and software environments.

The separation of switching hardware from the control software is fundamental to open programmability of networks, leading to abstractions and virtualisation of network infrastructure. A distributed CORBA-based control system is the lowest layer around each of the network entities, so that they can be realised as high level objects with well-defined interfaces. On top of this underlying architecture, applications for network control can be written to bind objects to describe connection management algorithms. The concept of open signalling is attractive to the military environment in which flexibility is extremely important in managing network resources. Mechanisms such as service prioritization and pre-emption can be utilised in the reservation of network resources, allowing dynamic control of military quality of service (QoS) for maximum military value.

This report reviews an open programmable platform for ATM networks, supporting the creation, deployment and management of network services. The feasibility of a connection management framework depends mainly on the performance of the CORBA mechanism, which inevitably generates binding overheads due to remote invocations. An implementation with reasonable performance may require certain criteria to be integrated into the design, including caching of network states, aggregation of access to the switch server object, and parallel processing of a single call request. Nevertheless, the effectiveness of these design criteria is still uncertain. Apart from the real time performance issues, reliability of signalling protocols is the most important, which requires careful design against all possible operating conditions and failures.

Although the use of binding abstractions seems to add a level of complexity to network control, the approach of open signalling is capable of supporting multiple mechanisms in connection management. As the new paradigm of distributed object computing becomes mature, the inherent complexity of open signalling will surely be reduced, thereby improving the overall performance in network control.

Authors

T. A. Au
Communications Division

Andrew Au is a Research Scientist in Network Architecture Group with interests in traffic control and performance guarantees for high speed networks. He has been investigating ATM signalling and communications issues in distributed computing environments.

Contents

1. INTRODUCTION 1

2. OPEN PROGRAMMABLE NETWORKING ENVIRONMENT 2
2.1 The Extended Reference Model 3
2.2 The Xbind Broadband Kernel 4
2.3 Connection Management Framework of Xbind 5
2.4 Connection Setup Procedure 6
2.5 Hierarchical Binding 7

3. PERFORMANCE OF CORBA 9
3.1 Binding Overheads of CORBA 9
3.2 Quality of Service Issues over Wide Area Networks 10

4. DESIGN OF THE CONNECTION MANAGEMENT SYSTEM 11
4.1 Design Features of Xbind 12
4.1.1 Caching of Network States 12
4.1.2 Aggregation of Access to the Switch Server Object 13
4.1.3 Parallel Processing of a Single Call Request 13
4.2 Reducing Remote Invocations for Better Performance 13

5. PERFORMANCE ISSUES OF OPEN SIGNALLING 14
5.1 Staleness of Network State Information 14
5.2 Effect of Processing a Single Connection Request in Parallel 15
5.3 Reliability 15

6. CONCLUSIONS 16

REFERENCES 18

APPENDIX - THE CORBA MECHANISMS FOR OBJECT ACTIVATION AND METHOD INVOCATION 19

1. Introduction

In recent years, the Common Object Request Broker Architecture (CORBA) created by the Object Management Group (OMG) has been recognised as an important enabling technology for the development of distributed object oriented applications. This trend can be witnessed by the fact that a number of industrial consortia have positioned CORBA at the core of their functional and software architectures. Examples are the Telecommunications Information Networking Architecture (TINA) Consortium, and the TeleManagement Forum[1]. CORBA is indeed an improvement over conventional remote procedure call (RPC) because the former supports object-oriented language features and more flexible communication mechanisms.

The flexibility of CORBA stems from its support for interoperability across heterogeneous hardware and software environments. An object request broker (ORB) allows objects to communicate with one another on a network. In particular, CORBA allows the interoperability between programming languages, applications, operating systems, and networking technologies. CORBA is designed to enhance distributed applications by automating many common networking tasks such as object location, parameter marshalling, request demultiplexing, and object activation. Building and implementing distributed applications means taking both the network and the software applications into consideration. Three important types of resources supporting distributed applications are communications, processing, and storage. These resources are integrated in the distributed object paradigm to constitute the end-to-end quality of service (QoS) requirements, facilitating the development of distributed applications. CORBA enables complex distributed and concurrent applications to be developed more rapidly and correctly, allowing seamless integration of legacy operational and support systems.

The key intelligence of an ATM network lies in its signalling system, providing the ability to establish, maintain and release connections between users and network nodes for network services. Essentially, the effectiveness of a signalling system strongly determines the total performance of the network. In particular, selecting a path involves QoS routing for each connection to satisfy diverse performance requirements and optimise resource usage. However, the path selection scheme should not consume excessive bandwidth, memory, and processing resources, so as to support high throughput and low delay in establishing connections in large ATM networks.

Recently, considerable attention has been drawn to the approach of open signalling for the support of multimedia applications across broadband networks. Based on a clear separation between switching hardware and the software used for network control, this concept creates an open architecture that provides a software layer around each of the network entities, such as switches, and multimedia devices. All of these resources are realised as high level objects with well-defined software interfaces, which can be

[1] Formerly the Network Management Forum (NMF).

used to model the entire network. Applications for network control can be written to bind objects on top of this underlying architecture. For instance, high-level language constructs can be produced to describe connection management algorithms operating on these objects. In this object-oriented approach of network control, the binding operations of objects contain a much lower level complexity than the standards based approach. General connection management architecture can be created whose lowest layer is a distributed CORBA-based control system, providing the high-level communications facilities required by the binding model. Typical examples are the TINA and xbind architectures, which are an attempt to improve upon the current approaches to connectivity services in B-ISDN or ATM networks.

The separation of communications hardware from control software is fundamental to open programmability of networks, leading to network infrastructure abstractions and virtualisation through open programmable interfaces. Applying this concept of open signalling in the military environment can enhance the flexibility in supporting military QoS. Mechanisms such as prioritization and pre-emption can be utilised in the reservation of network resources, allowing graceful degradation of QoS of existing connections at high load. This greatly increases the value of military information spread across various networking technologies, thereby fulfilling the operational requirements of modern battlefield interoperability.

Following this introduction, Section 2 reviews a reference model for open programmable networking. An open programmable platform for ATM networks called xbind [1] is described, supporting the creation, deployment, and management of network services. Since CORBA is used as the distributed object environment for open signalling, Section 3 discusses the performance of CORBA in binding remote network control objects. Section 4 reviews the design criteria proposed to reduce binding overheads due to remote invocations of the connection management system. The performance issues of this approach to open signalling are discussed in Section 5. Conclusions are provided in Section 6.

2. Open Programmable Networking Environment

Traditionally, the service model of telecommunications networks is a two-tiered architecture, consisting of a user domain and a network domain. In this customer-provider model, users are located at the periphery of the system that access network services via a user-network interface (UNI). Within the network domain, interconnection between switches is achieved via a network-node interface (NNI). There is little distinction between network provider and service provider, since most useful services require the intimate network support available only through an NNI. The interface between the network and the service architecture is rigidly defined and cannot be replaced, modified, nor supplemented, hence defeating the potential diversity and flexibility of user level services and applications.

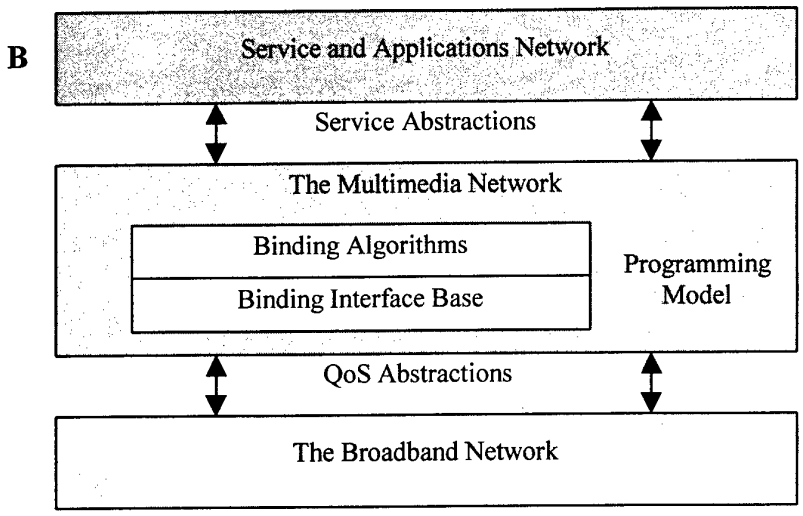


Figure 1: Overview of the RGB decomposition of the Extended Reference Model

The advent of the Internet technology demands an open environment for design, installation, and operation of network services. This is expected to offer users the flexibility of setting up their own service without unnecessary technical barriers due to the network domain.

2.1 The Extended Reference Model

Targeted towards multimedia services, the Extended Reference Model (XRM) models the communications architecture of networking and multimedia computing platforms, which better reflects the operating structure of the future communication services industry. The XRM consists of three components: the Broadband Network (the R-model), the Multimedia Network (the G-model), and the Services and Applications Network (the B-model), as illustrated in Figure 1 [1].

The broadband network (the R-model) represents the physical network that consists of switching and communication equipment, and multimedia end devices. Upon this physical infrastructure resides the multimedia network (the G-model), providing a programming model that allows service behaviour to be specified and executed. The primary function of the multimedia network is to provide the middleware support through a set of QoS abstractions. These QoS abstractions are implemented as a collection of distributed software entities with open interfaces, managing the states of local multimedia resources in the broadband network, including the devices, switches, links, processors, and their respective capacities. These interfaces are consolidated into the Binding Interface Base (BIB), as basic building blocks of a multimedia network.

The basic resource allocation and management services are realised as algorithms of the multimedia network operating on the BIB for the creation of simple point-to-point

connectivity with guaranteed service characteristics. These algorithms are native to the multimedia network with the goal of implementing a set of rudimentary communication services, collectively termed as the Broadband Kernel Services (BKS). Typically, functions such as connection management, routing, and admission control are required for communication services. The states of a service are represented by the service abstractions to be used by the services and applications network (the B-model) for managing and creating new services through dynamic composition and binding. By invoking the service abstractions, the services and applications network is able to assemble a set of independent network services to create yet higher consumer level services.

The XRM is based on a clear separation between switching hardware and the software used for network control. This concept establishes a new approach of open signalling for the support of multimedia applications, thereby creating an open architecture that provides a software layer around each of the network entities, including switches and multimedia devices. All of these resources that participate in the binding process are modelled as high-level objects with well-defined software interfaces. These objects are stored within the BIB, facilitating the construction of a model of the entire network. Applications for network control can be written to bind objects without the need to change the underlying architecture. High-level language constructs can be produced to describe connection management algorithms operating on these objects.

2.2 The Xbind Broadband Kernel

Based on the G-model of the XRM, an open programmable platform on ATM networks called xbind [1] has been developed by the COMET group at Columbia University, which supports the creation, deployment and management of networked multimedia services. It includes software components for implementing mechanisms for distributed resource allocation, broadband signalling, real-time switch control, and multimedia transport and device management. Xbind maintains a simplified BIB, prototypes of all broadband kernel services, and a simple teleconferencing service with QoS renegotiation capabilities. This framework can also be exploited to provide general network management and control in ATM networks.

In principle, IP based networks are ideal for the reliable transport of short messages with little or no call holding times, whereas connection-oriented ATM networks are highly suited for transport of streams with QoS guarantees and long call holding times. To exploit the advantages offered by IP and ATM technologies, the object-oriented signalling infrastructure of xbind is built on IP based channels while the transport mechanisms are native to the underlying ATM network. The object-binding framework is based on CORBA running over the TCP/IP protocol suite. All controllers of xbind are modelled as objects interacting through object invocations defined using the interface definition language (IDL). Overall, the signalling infrastructure is composed of a collection of ORBs that provide a homogeneous name space across a distributed set of heterogeneous platforms.

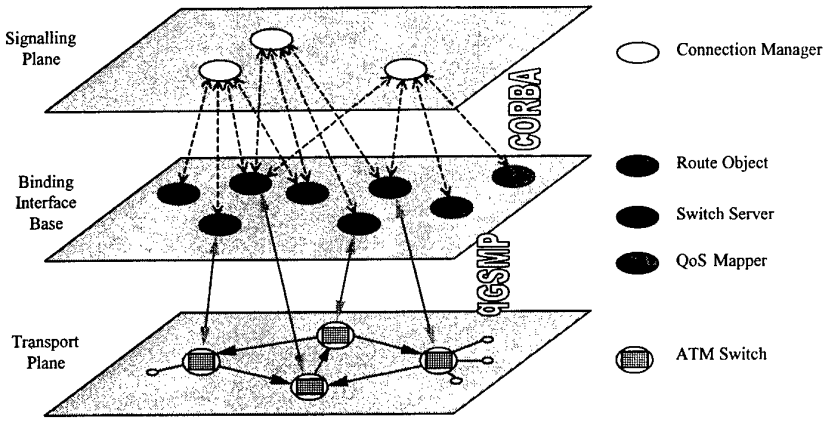


Figure 2: Object interaction model

Mapping of the states of the switch hardware into the software domain can be achieved with the QoS-extension of General Switch Management Protocol, referred to as qGSMP [2]. This is a standard interface providing a number of key features for development of switch control software. Included are the means of specifying QoS constraints, selecting scheduling and buffer management policies, and transferring schedulable regions. A schedulable region provides a unified notion of resource capacity abstraction at call level that defines traffic classes of customers according to their network requirements.

2.3 Connection Management Framework of Xbind

Connection management architectures are required to get access to interfaces that provide sufficient information about the network topology, the state of the network elements, and the current link utilisation. As the network operating under the xbind broadband kernel becomes programmable, various binding algorithms can be built on top of the information that the BIB holds. Since the BIB is populated with network control objects for controlling ATM switches, it is possible to construct connection management algorithms of much lower level complexity compared to current approaches such as the UNI and NNI specifications within the ATM Forum standards. The separation of the connectivity software from the switching hardware also facilitates the access to connectivity services by a third party, allowing, for example, third-party connection setup and leaf-initiated joins. In actual fact, the leaf-initiated join capability is defined in the UNI 4.0 specification, and the capability for third-party call setup will be an added feature for the next release of the UNI specification.

In support of the interoperability between network entities in a multivendor environment, the lowest layer of the connection management architecture in xbind is based on CORBA, which is used as a trader for matching objects requesting and consuming services exported from other objects. Figure 2 shows the logical object interaction model of the connection management architecture in a plane structure [3].

The transport plane contains a set of interconnected ATM switches for the transport of multimedia flows. In the middle plane, the binding architecture exports a set of BIB interfaces for controlling and monitoring network resources. Three object types are required to perform tasks for connection control, namely, QoS Mapper, Route Object, and Switch Server. QoS Mapper provides mapping of user-level service abstractions to QoS abstractions for network resources. For example, QoS abstractions for video services are specified in terms of frame rate and frame loss of video streams, which are translated to QoS abstractions using specific cell loss and cell delay requirements for each traffic class. Route Object selects a route between two endpoints in the network, and the Switch Server provides generic, hardware independent interfaces for manipulating resources on an ATM switch using qGSMP. Running on this set of BIB interfaces, connection management algorithms reside in the signalling plane as CORBA objects, which communicate over an IP based network for the transport of relatively short control messages.

2.4 Connection Setup Procedure

The order of executions during call setup requests is illustrated in Figure 3. The client application can reside on one of the two hosts, or any other machine for third party connection setup. In step 1, a client application program initiates a request to the Connection Manager to set up a connection between two hosts. QoS mapping is processed in step 2 to translate the user-level QoS to network-level QoS. Based on the QoS requirements, step 3 selects a path in the network from the Route Object to connect these two endpoints. Resource reservation is performed in step 4-13 to the two hosts and each of the Switch Server objects. In step 14, the Connection Manager returns the status of the connection setup to the client program.

The connection management framework does not specify the actual location of the objects, leaving this decision to actual implementations. At one extreme, all objects could be instantiated in a central computer, while communicating with switches by means of a switch control interface such as Simple Network Management Protocol (SNMP). At the other extreme, an object representing a physical entity might run directly on the entity represented. In the latter case, the performance of the connection management depends entirely on the processing power of switch hardware.

The execution of a connection setup request emulates the procedure of the UNI and the Private Network-to-Network Interface (PNNI) specifications if the resource reservation is performed sequentially along the path from source host to destination host. On the other hand, the Connection Manager can performed the resource reservation in a single step if requests are issued independently and in parallel to the two hosts and all the switches along the selected path. Nevertheless, the process of setting up a virtual path identifier (VPI) and a virtual circuit identifier (VCI) in the routing table requires two phases. In phase one, an output VPI/VCI pair is obtained in steps 4-8 from the output port of each of the switches located in the path of the call. Phase two commits the

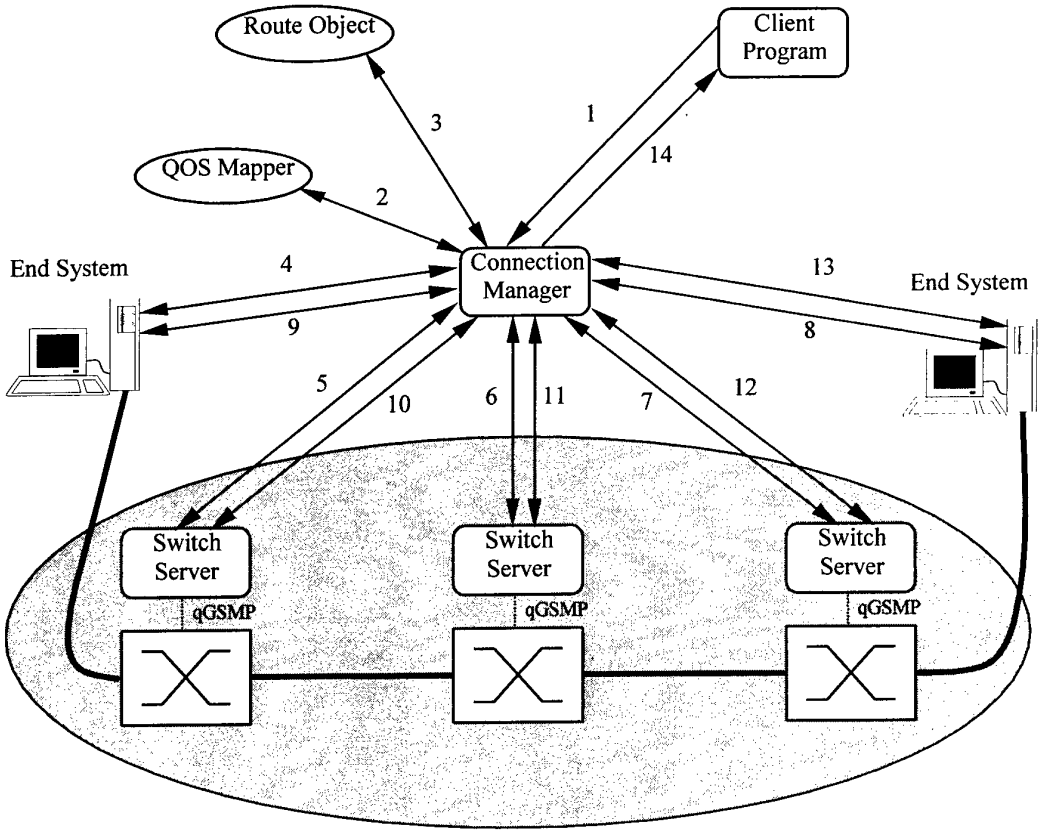


Figure 3: Execution of a connection setup request

channel by mapping the output VPI/VCI pair of the upstream switch into the input VPI/VCI pair of the downstream switch through steps 9-13.

2.5 Hierarchical Binding

Xbind is the lowest layer of general connection management architecture, facilitating the implementation of any applications over the BIB. It allows the integration of arbitrary end-to-end bindings and the manipulation of its underlying network resources. In particular, routing algorithms can be designed in one of two ways: peer-to-peer relationship, or client-server relationship. The concept of connectivity services in the UNI and PNNI specifications is typical of a peer-to-peer relationship, in which information is passed to network entities one by one, whereas a client-server relationship is adopted in the connection management architecture of TINA. TINA is expected to enable the network to support a multiplicity of call models and signalling protocols. By defining high-level building blocks and the models of interactions between them, TINA achieves greater modularity than the current ATM Forum standards. In this hierarchical binding, binding tasks are effectively delegated to

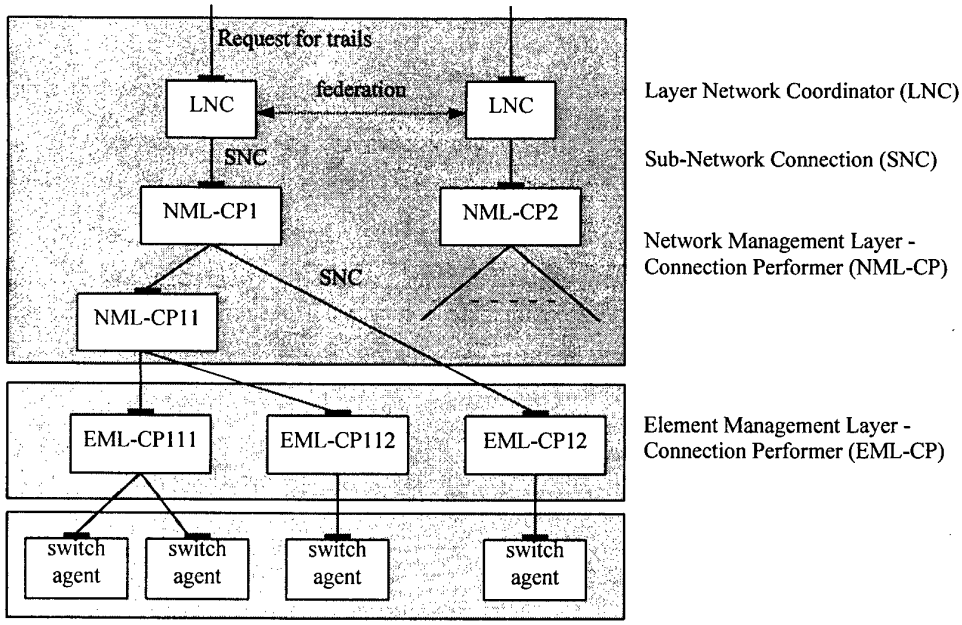


Figure 4: TINA network control structure

various domains including the end systems and the associated networks, thus enhancing scalability.

The connection management architecture of TINA defines a hierarchy of connection performers over a collection of interconnected networks until open interfaces to switch resources are reached at the lower level [4]. As shown in Figure 4 [5], a network hierarchy is formed through parent-child relationship between subnetworks, such that a child subnetwork is represented in the parent as a single node. By maintaining the hierarchically aggregated network state information, the requests for connection and disconnection are passed through the appropriate controlling object associated with the subnetwork to which the request is made. Routing in TINA is achieved by the decomposition of trails into smaller requests to connection performers down the hierarchy, until the switch agents in the fabric to set up their part of the trail. If an individual connection request is not successful, a crankback mechanism can be used to find an alternative path. Once the connection request reaches the bottom layer, the creation of connection through single network elements such as switches proceeds independently and in parallel. Therefore, the decision whether to accept or deny the request is made centrally within that subnetwork. In comparison, the hierarchical aggregation of the PNNI is very similar but routing is source based and setup initiation is signalled along the chosen route.

3. Performance of CORBA

Since CORBA provides the high-level communications facilities for the binding operations of network control objects, its performance has a significant impact on the overall efficiency of general connection management architecture. CORBA provides programmers with location and distribution transparency, along with hardware, operating system and programming language transparency. The basic mechanisms of CORBA for object activation and method invocation are described in the Appendix, which inevitably generate associated overheads for the benefits of transparencies.

In CORBA, distributed objects behave like collocated objects, as if they resided in the same memory space, making remote requests appear to be processed through direct function calls. The CORBA IDL acts as a language bridge, whose language independence feature allows the choice of language for a component without impacting others. Based on the Berkeley software distribution (BSD) socket programming interface, CORBA is used as a high-level network programming interface for building a large distributed system. In recent years, CORBA has become a well-established and widely adopted standard, and well suited to provide a framework for flexible and transparent communication between distributed objects in heterogeneous computing environments.

3.1 Binding Overheads of CORBA

Unfortunately, the benefits of CORBA do come at a price. The trade-offs for the properties of transparency in CORBA are binding overheads and remote operation costs. Figure 5 shows the general path that CORBA implementations use to transmit requests from client to server for remote operation invocations [6]. The CORBA overheads can be attributed to many factors: presentation layer conversions and data copying, server demultiplexing, and buffering for network reads and writes. To achieve higher throughput, the primary areas that must be optimised include reads/writes, presentation layer conversions and data copying. Indeed, a reasonable percentage of the execution time is spent in memory copy operations. In any case, the number of reads and writes can be reduced by increasing the sender buffer size to match the maximum transfer unit (MTU) of the underlying network, thereby enhancing the throughput [7].

Among others, the presentation layer is in fact a major bottleneck in high-performance communication systems. This layer transforms all typed data objects from higher-level representations to one suitable for transmission over the connection to the target object, and vice versa (called marshalling and demarshalling respectively). This transformation is processed by client-side stubs and server-side skeletons. Further, the performance of CORBA varies for transferring different CORBA IDL types. For example, the use of IDL `strings` and `structs` carries some performance penalties compared to the use of `sequences` and `chars` [8]. The `structs` data type allows

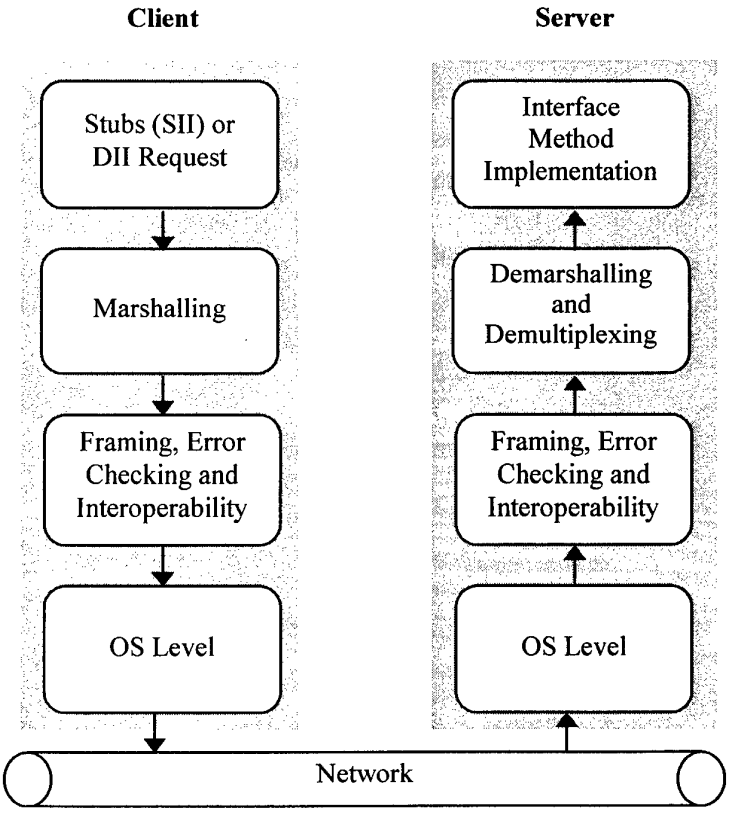


Figure 5: General path of CORBA requests

related data types to be packaged together, whose implementation has to spend a significant amount of time creating a request, populating it with parameters and marshalling data. The server also has to spend a significant amount of time performing reads and demarshalling the data. Hence, the overhead in marshalling and demarshalling constitutes a significant performance drop. Of all the CORBA data types, the `chars` and `octets` perform the best because the presentation layer conversions and data typing can be minimised [7].

3.2 Quality of Service Issues over Wide Area Networks

The performance of CORBA seems acceptable on local area networks, but the primary drawback is exposed when its potential over wide area networks is evaluated. Besides, due to the lack of scalability, CORBA is unlikely to support a vast number of objects distributed among numerous machines over a wide area network. With increasing network speeds, the performance of the CORBA implementations actually becomes aggravated compared to low-level interfaces like sockets [6]. Indeed, distributed applications are increasingly being deployed across wide area networks whose environments are dynamic and hostile. These applications are more difficult to develop and maintain than those distributed across a local area network. To help simplify the

development of wide-area distributed applications, CORBA should be developed to provide application-level quality of service [9].

CORBA is a high-level network-programming interface that offers many advantages over low-level interfaces such as the BSD sockets. Among these advantages are extensibility, maintainability, and reusability. However, in comparison with sockets, Orbix (the CORBA implementation of Iona Technologies) exhibits a drop in performance of up to a factor of three, depending on the message and socket buffer sizes [8]. High-level interfaces are obviously less efficient than the low-level interfaces due to the layers of binding software, especially on high-speed networks. Nevertheless, users may be willing to accept a certain performance penalty given all the benefits they are gaining from using these high-level interfaces.

The inefficiency of contemporary CORBA implementations due to excessive marshalling, data copying, demultiplexing, and memory management overheads is a serious problem for mission-critical applications such as process control systems, medical imaging, and video conferencing. The current CORBA standard is not able to support real-time constraints for these applications. Only by applying efficient optimisations to the CORBA stubs and skeletons can these performance bottlenecks be alleviated. Further, real-time applications require a CORBA run-time system that supports enforcement of the timing constraints. The same requirements apply to the underlying operating environment supporting distributed applications with real-time constraints. This includes the operating systems on both the client and server nodes, and the network that they use to communicate. Meanwhile, developers may be forced to choose lower-level mechanisms (like sockets) to achieve the necessary transfer rates, thereby increasing development effort and reducing system reliability, flexibility, and reuse.

4. Design of the Connection Management System

Using CORBA as the control mechanism between the Connection Manager and switches inevitably involves communication costs, affecting the overall performance of the Connection Manager. A client program makes a connection request to the Connection Manager based on a particular traffic type and a set of desired QoS parameters. If the connection management architecture of the network is based on TINA, this single request is hierarchically partitioned across the network until the decision is taken locally at each switch involved in the path. Call admission control (CAC) and resource reservation decisions are made at each switch locally. These strategies however may differ from switch to switch.

4.1 Design Features of Xbind

For the support of a large number of network users, it is essential that a connection management system be designed to exhibit high performance. In a distributed environment like CORBA, the most expensive operations are remote object invocations. In fact, the vast majority of the remote operations contain small arguments. These remote invocations contribute the bulk of the latency in call processing. Hence, the following design criteria [3] have been adopted into the implementation of xbind, to reduce the overheads due to remote invocations: caching of network states, aggregation of access to the Switch Server object, parallel processing of a single call request:

4.1.1 Caching of Network States

To minimise the number of remote invocations on objects for connection control (ie, the QoS Mapper, the Route Object, and the Switch Server), part or all of its states can be stored or cached in the Connection Manager. These network states include QoS mapping, route, input or output switching identifiers, bandwidth and buffer resources, and existing connection states.

QoS Mapper is relatively static, which can be collocated in the same address space as the Connection Manager. In this way, QoS mapping becomes a local invocation rather than a remote invocation. Caching routes reduces the remote invocation to the Route Object each time a setup request for a specific source-destination pair arrives at the Connection Manager. Its effectiveness depends on the patterns of call requests that, in turn, determine the source-destination pairs to be cached. Higher frequency of repeated call requests for the same source-destination pair reduces the number of remote invocations for route selection; however these cached routes may soon become invalid in a network operating at high load.

Setting up a connection in a switch involves mapping the output VPI/VCI pair of the upstream switch to the input VPI/VCI pair of the downstream switch. Two phases are required to perform reservation of VPI/VCI values in the routing tables, unless the Connection Manager first reserves or prefetches a set of available output VPI/VCI pairs from each of the Switch Servers. In this way, the Connection Manager simply looks for an available VPI/VCI pair in its name space cache. If an available output VPI/VCI pair is found (a cache hit) for each switch on the path of the call, the channel reservation process can be performed in a single step; otherwise the normal two-step operation is performed for all switches with cache misses.

Caching of bandwidth and buffer resource is similar in nature to VPI/VCI name space caching. By caching existing connection states locally in the Connection Manager, the state of the existing connections can be accessed using local object invocations. This information is particularly useful during QoS renegotiations for faster connection modifications.

4.1.2 Aggregation of Access to the Switch Server Object

In order to increase the throughput of the system, access to remote objects can be aggregated as much as possible. Instead of making one individual invocation to the Switch Server per request, multiple requests are combined into a single remote invocation. The delivery of requests to the Switch Server is therefore delayed, reducing the total number of remote invocations performed by the Connection Manager. The delay introduced by waiting inevitably increases the expected setup time, which, nevertheless, can be outweighed by the reduced number of remote invocations at higher load.

4.1.3 Parallel Processing of a Single Call Request

Upon receiving a request to set up a connection from a client application, a Connection Manager creates an instance of connection object to which the task is designated. A routing agent is consulted to determine the best path between the source and the destination. Normally, the connection is established by performing the set up process sequentially on every switch along the selected path. Alternatively, the system can be designed to run with a maximum amount of parallel processing to reduce the accumulated latency due to remote invocation. Processing a single connection request concurrently at each intermediate switch facilitates fast connection establishment for large networks. Parallel execution of connection establishment allows the resource reservation at intermediate switches be verified in parallel, instead of sequentially on a switch-by-switch basis as in the PNNI specification. In this case, two phases are required to set up the connection. The first phase reserves the output VPI/VCI pair of each of the switches along the path, and the second phase maps the input VPI/VCI pair to the output VPI/VCI pair of the upstream switch. This mapping is written into the routing table of the switches in parallel and the connection is committed.

To reduce the latency of setting up a connection even further, a connection request can be processed in one phase by centralising the allocation of VPI/VCI pairs. An output VPI/VCI pair of a switch is generated at random within a Connection Manager, which is mapped to the input VPI/VCI pair of the downstream switch in parallel in a single step. However, this one-phase approach is degenerated to a two-phase algorithm if the proposed VPI/VCI pair is already in use.

4.2 Reducing Remote Invocations for Better Performance

By integrating these design criteria into the implementation of the Connection Manager, most of the invocations are transformed from remote to local calls, except for one remote access to the Switch Server. The requests to the Switch Server are not delivered immediately. Instead, these messages are put in a message aggregation module and are only sent when either the number of messages reaches a particular threshold, or when a time-out occurs. Caching and request aggregation schemes reduce

the number of remote accesses, thus improving the latency and throughput of call processing. Besides, parallel processing of a single call request at multiple switches along the path is capable to enhance the performance of call processing.

Results show that an implementation of this connection management model provides reasonably good performance with low latency and high throughput [3]. Nevertheless, faster computing power is definitely required in the Connection Manager to execute different functions of call processing for a reasonable performance.

5. Performance Issues of Open Signalling

In this section, we discuss the effectiveness of the design features of xbind in reducing the number of remote operation invocations, especially on the overall performance of the connection management system.

5.1 Staleness of Network State Information

Caching of network states reduces the number of remote invocations and thereby suppresses the latency as much as possible. Caching of the QoS Mapper in the Connection Manager is most justifiable on the grounds of the relatively static mapping of user-level QoS to network-level QoS. Importantly, multiple Connection Managers may exist in a network accessing the same objects, each of which is allowed to function independently without regard to one another. Thus, caching all network information in one Connection Manager does not guarantee an exclusive right to manipulate on the network states. Further, caching of state information from the Route Object and the Switch Server is a source of inaccuracy that may have a significant impact on the decision of resource reservation. Typically, the one-phase approach may effectively become the two-phase approach due to the staleness of the switching identifiers. The stale network information increases the uncertainty about the actual state of the network, which affects the success rate of establishing new connections.

In particular, the Route Object creates the network topology from current network states from which the routes of a source-destination pair are determined. The utilisation of caching route information depends on the frequency of connection establishment and termination in the network. At high load of connection requests, changing network states may drastically modify the selection from the route determination process, resulting in frequent crankbacks to establish incoming connections. This phenomenon increases the number of failures to set up new connections, while consuming significant resources inside the network.

Besides the generation of excessive overheads in establishing connections at high load, the stale information about the switches and existing connections may lead to an avalanche of state inaccuracy in a network hierarchy. To ensure scalability in large

networks, the state information is often aggregated and disseminated up a network hierarchy. In this case, the staleness of the network states further deteriorates the inherent inaccuracy of higher abstraction, contributing to the overall inaccuracy of network signalling.

5.2 Effect of Processing a Single Connection Request in Parallel

The parallel algorithms in processing connection requests do not scale as well as the switch-by-switch algorithm. A task can benefit from parallel execution only if the incurred communication and computation overhead is relatively small. In our case, the overhead for a connection object is to coordinate all the connection agents on the connection path. The performance could further worsen if the average number of switches on a path is large such as in wide area networks. The additional overhead in coordinating the switches to establish a connection in parallel makes its scalability questionable.

Further, a comparative study [10] shows that the performance of setting up connections in parallel is not always better than that of the traditional sequential approach, since more individual request messages are generated in parallel for one single connection request. Overall, the one-phase algorithm always performs better than the two-phase algorithm due to the savings in remote accessing of VPI/VCI parameters. At very low traffic load, the approach of parallel execution is shown to yield lower mean call setup delay than the sequential approach [11]. However, as the rate of incoming connection requests increases, the setup time of the two-phase algorithm degenerates rather fast. At higher load, the performance of both one-phase and two-phase algorithms is worse than that of the sequential approach because of the significantly higher execution overhead at the Connection Manager. In any case, hybrid solutions that utilise the best of the parallel algorithms together with the sequential approach may be able to provide a balance of the overall performance.

5.3 Reliability

Despite its enormous size and complexity, the public switched telephone network (PSTN) is perhaps the largest and most complex distributed system, which is also among the most reliable. Likewise, an ATM network operating over wide area is expected to contain a large number of switches, including redundant hardware and extensive self-checking and recovery software. As such, the seemingly trivial task of connecting one point to another requires some of the most complex and sophisticated computing systems in existence.

The reliability of large distributed systems relies on all the software, hardware, and human operators and technicians to function correctly. Failure of any one of these elements can disrupt or bring down an entire system. Issues of reliability and real time performance are the most important in the signalling system of such a network. Indeed, much of the research of switch manufacturers is focused on developing highly

reliable systems, expecting their switches to experience high reliability. For instance, the failure rate of AT&T telephone switches is 5.7×10^{-6}, which is not more than two hours of failure in 40 years [12]. The software in switches demonstrates low failure rate using the best development practices. It is not surprising that software for a switch with even a relatively small set of features may comprise several million lines of code. Typically, half of the software in telephone switches is devoted to error detection and correction.

Traditional signalling techniques such as the UNI and PNNI specifications are often criticised to be old-fashioned [3,4,5], whose entire protocols involve a huge amount of code running on the switches and the end systems. While this might be true, its reliability is something that open signalling cannot match without careful considerations being put into its signalling procedures. To facilitate deployment for commercial use, the connection management framework must be thoroughly tested for its robustness under all kinds of operating conditions. Building the protocol for open signalling with robust logic to recover from all possibilities of human actions may eventually multiply the amount of basic code. This addition of software contains extensive self-checking and recovery functions, inevitably affecting the total response time and therefore the latency in establishing connections.

6. Conclusions

ATM switches usually come with a control processor. Although it is possible to execute signalling functionality on the processor, it could be easily overloaded as the signalling volume goes up. The concept of open signalling advocates the opening up of signalling platform so new network algorithms can be readily deployed by third parties. This framework considerably eases the implementation of the signalling protocols required to establish new services, which can be viewed as application programs running over the network. Also, additional processing capability can be easily added as connection volume increases or as signalling software becomes more advanced. In contrary, the traditional signalling framework using the UNI and PNNI specifications is closed, which is meant not to be controllable by users. The concept of open signalling is rather attractive to the military environment in which flexibility is extremely important in managing network resources. By monitoring new connection requests and existing connections, mechanisms such as service prioritization and pre-emption can be used to optimise the value of military information without affecting the overall operational requirements.

We have discussed the design of an open programmable networking environment for ATM networks, the architecture of which consists of a number of CORBA servers providing connection management functionality in both the end systems and the network. CORBA provides a technology-independent object-based computational model in which distributed objects can be invoked with full location transparency.

Other benefits of using CORBA include distribution transparency, programming language independence, and platform independence. The approach of using CORBA to handle connection management provides an extremely open, standards-based, extensible, and distributed network management system. The use of CORBA interfaces facilitates easy and seamless integration of the connection management framework into other systems. Unfortunately, the benefits of CORBA come at a price. Binding overheads are generated along the path to transmit requests from client to server for remote operation invocations. Reducing these overheads means trading off some of the benefits that CORBA offers.

In addition to a suitable architectural design, an implementation with reasonable performance may require certain criteria to be integrated into the design. The design features of the xbind implementation are caching of network states, aggregation of access to the Switch Server object, and parallel processing of a single call request. In effect, most of the invocations are transformed from remote to local calls, except for one remote access to the Switch Server, thereby providing reasonably good performance with low latency and high throughput. Nevertheless, the effectiveness of these design criteria is still uncertain. The stale network information increases the inaccuracy about the actual state of the network, which affects the success rate of establishing new connections, especially at high load. In large networks, this staleness may further deteriorate the inherent inaccuracy of higher abstraction due to the network hierarchy. Besides, the scalability of processing connection requests in parallel is still questionable. Its performance is not always better than that of the traditional sequential approach.

Apart from the real time performance at both the cell-level and call-level, reliability is undoubtedly the most important in the signalling system of large networks. High reliability of signalling protocols is attributed to careful design against all possible operating conditions and failures. This may eventually complicate the basic signalling procedures, adding extensive self-checking and recovery functions.

The approach of open signalling has demonstrated its feasibility in supporting multiple mechanisms of network control. While lowering the complexity in programming connection management algorithms, the use of binding abstractions seems to add a complexity to network control. However, there are a number of distributed object environments, and CORBA is not the only option. As this new paradigm matures in the future, the complexity of open signalling will surely be reduced, improving the overall performance in connection management.

References

[1] A. A. Lazar, "Programming Telecommunication Networks," IEEE Network, Sep/Oct 1997, p. 8-18.

[2] C. M. Adam, A. A. Lazar, and M. Nandikesan, "QOS Extension to GSMP," OPENSIG Workshop on Open Signaling for ATM, Internet and Mobile Networks, UK, April 17-18, 1997.

[3] Mun Choon Chan, "Architecting the Control Infrastructure of Broadband Networks," Ph.D. Thesis, Columbia University, 1997.

[4] H. Oliver, C. Edwards, F. D. Tran, J.-B. Stefani, and D. Hutchison, "Supporting Real-Time Multimedia Applications with Distributed Object Controlled Networks," Proceedings of the 1st IEEE Symposium on Object-Oriented Real-Time Distributed Computing, Kyoto, Japan, 20-22 April 1998.

[5] H. Oliver, S. Brandt, A. Thomas, and N. Charton, "Network Control as a Distributed Object Application," Distributed Systems Engineering, vol. 5, no. 1, March 1998, p. 19-28.

[6] A. S. Gokhale, and D. C. Schmidt, "Evaluating CORBA Latency and Scalability Over High-Speed ATM Networks," IEEE Transactions on Computers, April 1998.

[7] A. Gokhale, and D. C. Schmidt, "The Performance of the CORBA Dynamic Invocation Interface and Dynamic Skeleton Interface over High-Speed ATM Networks," IEEE GLOBECOM, Nov 1996.

[8] R. A. Fatoohi, "Performance Evaluation of Communication Software Systems for Distributed Computing," Distributed Systems Engineering, 4(1997), p. 169-175.

[9] D. E. Bakken, R. E. Schantz, and J. A. Zinky, "QoS Issues for Wide-Area CORBA-Based Object Systems," Second Workshop on Object-Oriented Real-Time Dependable Systems, 1996, p. 110-112.

[10] I. M.-C. Tam, W. Wang, and A. Lazar, "A Comparative Study of Connection Setup on a Concurrent Connection Management Platform," IEEE Open Architectures and Network Programming, 1998, p. 14-24.

[11] R.-H. Hwang, J. F. Kurose, and D. Towsley, "The Effect of Processing Delay and QOS Requirements in High Speed Networks," Technical Report 91-52, Department of Computer Science, University of Massachusetts at Amherst, 1991.

[12] D. R. Kuhn, "Sources of Failure in the Public Switched Telephone Network," IEEE Computer, vol. 30, no. 4, April, 1997, p. 31-36.

[13] T. A. Au, "A Primer of CORBA: A Framework for Distributed Applications in Defence," DSTO Report DSTO-GD-0192, March 1999.

[14] P. E. Chung, Y. Huang, S. Yajnik, D. Liang, J. C. Shih, C.-Y. Wang, and Y.-M. Wang, "DCOM and CORBA Side by Side, Step by Step, and Layer by Layer," the C++ Report, Jan 1998.

Appendix - The CORBA Mechanisms for Object Activation and Method Invocation

The Common Object Request Broker Architecture (CORBA) [13] offers a distributed object infrastructure for transparent activation and accessing of remote objects. CORBA is based on an object-oriented distributed model and designed for distributed computing using a client-server model. Essentially, CORBA is a middleware that acts as a software layer situated between the application and operating system to facilitate the design and implementation of client-server systems in a heterogeneous environment. The interactions between a client process and an object server are implemented as object-oriented remote procedure call (RPC) communications. The Object Request Broker (ORB) is the object bus over which objects interact transparently with other objects located locally or remotely. The ORB is responsible for all the mechanisms required to find the object implementation, transmit the request, and the response if any back to the client.

A.1 RPC Structure

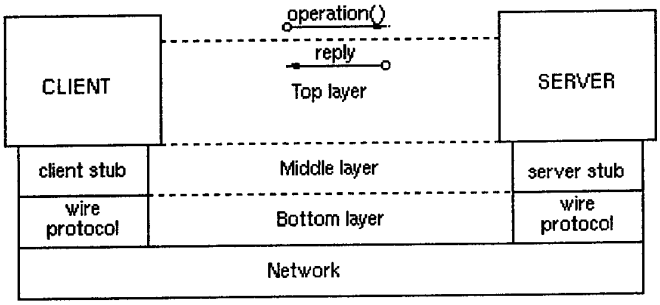


Figure 6: RPC structure

As extensions to local procedure calls, remote procedure calls (RPCs) allow a client to invoke a remote server transparently, as if the application resided on the same system. RPC is an asymmetric type of communication that is based on a client-server model. The RPC structure can be partitioned into three layers [14], as shown in Figure 6. The top layer is the basic programming architecture visible to the developers of the client and server programs. The middle layer is the remoting architecture, where the object references are made transparent across different processes. At the bottom layer, the transmission or wire protocol architecture further extends the remoting architecture to work across different machines.

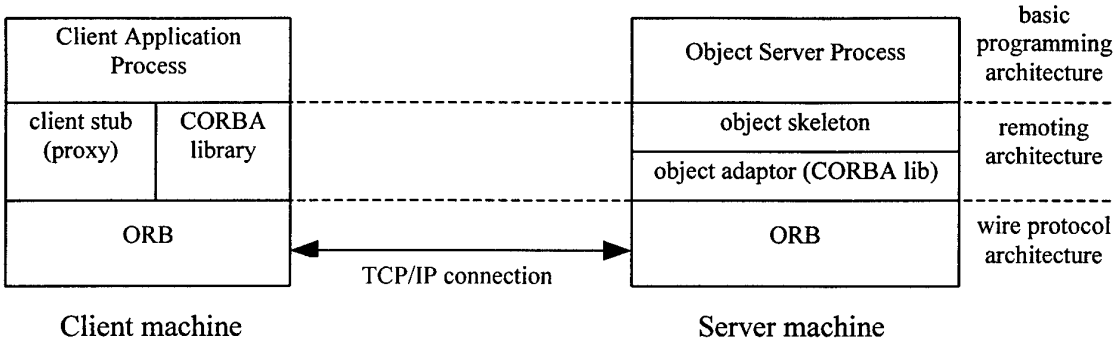


Figure 7: CORBA overall architecture

A client invokes a remote operation to a server by making a call to the client stub. The client then waits for an action to occur and receives a reply about the success or failure of the request. In fact, the stub packs the call parameters into a request message, and invokes a wire protocol to transfer the message to the server. The message is delivered to the server stub, which then unpacks the request message and calls the actual operation on the object. The client stub and the server stub are called the stub and the skeleton in CORBA, respectively.

A.2 CORBA Overall Architecture

The RPC structure is a precursor of the CORBA, and Figure 7 illustrates the overall architecture of CORBA [14]. At the top layer, the exact connection mechanism between the client and the server is totally hidden from the application programmers, as if the client and the server programs reside in the same address space on the same machine. The client can activate a server object by invoking any method on its object reference.

The middle layer provides the necessary infrastructure, so that the client and the server appear to be in the same address space. The stub and skeleton are used to hide the details of the underlying ORB from the application, making remote invocation look similar to local invocation. When the client invokes a remote operation it is invoking the operation of that name provided by the client stub. This routine typically resides in the same process as the stub and the client, perhaps linked in as a run-time library. In particular, the middle layer involves marshalling and demarshalling for sending data across different address spaces. Marshalling bundles the call parameters from the client or return values from the server into a standard format for transmission. Demarshalling performs the reverse operation, which converts the standard format back to an appropriate data presentation in the address space of a receiving process.

The bottom layer specifies the wire protocol for transmission between the client and the server running on different machines. The ORB core is used to locate the remote object, prepare it to receive the request, and finally hand it the request in the form of a buffer so that it can execute the appropriate operation. On the server side, the

arrangement of libraries and processes at the remote object is similar to that at the client. Typically, the remote object is contained in a process linked with the IDL skeleton, the basic object adaptor (BOA), and some routines provided by the ORB core.

ORBs can be built using a variety of different protocols to communicate, typically using the User Datagram Protocol (UDP) of the TCP/IP protocol suite. The wire protocol for inter-ORB communication between ORBs from the same vendor is vendor dependent. For general ORB interoperability, the CORBA 2.0 specification includes a general inter-ORB protocol (GIOP) for ORB-to-ORB interactions, in which a set of message formats and common data representations are specified. The GIOP on TCP/IP is known as the Internet inter-ORB protocol (IIOP) for interoperability among TCP/IP based ORBs.

A.3 Object activation

To invoke operations on an object, the client must first obtain its object reference. This is usually accomplished through a naming service, or by using a bind method and specifying the object type. For example, Iona Orbix and Visigenic Visibroker both provide a static `_bind()` operation that the client can use to activate a server object and obtain its object reference.

When the client invokes the bind method in the stub, the stub first checks its proxy object table to see if it already has an object reference for the remote operation. This proxy object table maintains a run-time table of all valid object references on the client side. If the client stub does not have a valid object reference for the remote operation, the task is delegated to the client-side ORB. The ORB then consults a locater file to choose an object implementation that offers the requested interface.

The object implementation may reside in the same process as the client program or it may reside in a separate process called a server. If the object implementation is located in the same process as the client, a proxy object is not created. Instead, a pointer to the object implementation itself is returned to the client program. Otherwise, a request is sent to the server-side ORB, typically via the TCP/IP protocol suite.

On the server side, the BOA is primarily responsible for activating and deactivating the object implementation. The server-side ORB first consults the implementation repository to map the requested interface to its server path name. The implementation repository also provides platform-specific information to be used by the BOA to start up the object server. If the server that implements the requested object is not currently executing, the BOA then activates the appropriate server, which in turn instantiates all supported objects so that it can receive requests from the client program. The BOA also creates a socket endpoint, and assigns a unique implementation-specific object reference identifier to the requested object. Subsequently, a new object reference is bound to the requested object, containing the interface name, the implementation name, the reference identifier, and the endpoint address. If the IIOP is the underlying protocol for communication between the client and the server, the server must generate

an interoperable object reference (IOR) from the object reference whenever the object reference is passed across ORBs.

Eventually, the ORB returns the object reference for the requested object back to the client side. A proxy object is created in the stub and registered in the proxy object table with its corresponding object reference. Further, the proxy extracts the endpoint address and establishes a socket connection to the server. Once the connection is successfully established, the client is able to invoke methods on the proxy object, which in turn, interacts with the server object. In effect, the client's method invocations are translated into operation requests that are sent to the server object.

A.4 Method invocation

When the client invokes the remote operation on the proxy object, the IDL stub obtains a buffer and marshals into the buffer the operation name and the necessary parameters. The common data representation (CDR) format is required for GIOP message transfer. The buffer is handed to the underlying ORB, together with the object reference. The task of the ORB core on the client side is simply to locate the IP address of the remote host and the port number on which the server-side ORB is listening. Subsequently, the client-side ORB sends the buffer to the target server through the established socket connection. The client-side and server-side ORBs need to deal with any failures that may occur, such as lost or duplicated packets, giving the illusion of reliable communication.

After the server-side ORB has correctly received the buffer, the object reference is demarshalled to determine which object should receive it. If the object is not resident in the process, an error is returned to the client-side ORB. Otherwise, the remainder of the buffer is passed to the IDL skeleton for the object. The IDL skeleton demarshals the operation name, and the parameters for the operation. It then invokes the operation of the object, passing it these parameters.

Once the operation has executed, control returns to the IDL skeleton. The events that occur to send the results back to the client are similar to those that occur to send a request to a remote object. Specifically, the IDL skeleton obtains a buffer and marshals any results that are returned from the operation into the buffer. Likewise, the CDR format is used for GIOP message exchange. The buffer is then passed to the ORB core to be sent to the client. The ORB core needs to determine the return IP address and UDP port to which the buffer is sent. These are either contained as a field in the object reference received as part of the original request, or sent as a separate parameter.

Eventually, the reply buffer is returned to the client-side ORB. It checks that the client is indeed present in the process and passes the reply buffer to the appropriate IDL stub. The proxy object then demarshals the results, checks for exceptions, and returns them to the client program.

DISTRIBUTION LIST

A Preliminary Study of Open Signalling for ATM Networks

T. A. Au

AUSTRALIA

DEFENCE ORGANISATION

DGC3ID

S&T Program

Chief Defence Scientist
FAS Science Policy
AS Science Corporate Management
} shared copy

Director General Science Policy Development
Counsellor Defence Science, London (Doc Data Sheet)
Counsellor Defence Science, Washington (Doc Data Sheet)
Scientific Adviser to MRDC Thailand (Doc Data Sheet)
Scientific Adviser Policy and Command
Navy Scientific Adviser (Doc Data Sheet and distribution list only)
Scientific Adviser - Army (Doc Data Sheet and distribution list only)
Air Force Scientific Adviser
Director Trials

Aeronautical and Maritime Research Laboratory

Director

Electronics and Surveillance Research Laboratory

Director
Chief of Communications Division
Research Leader Military Information Networks
Head Network Architecture Group
Author: T. A. Au

DSTO Library

Library Fishermens Bend
Library Maribyrnong
Library Salisbury (2 copies)
Australian Archives
Library, MOD, Pyrmont (Doc Data sheet only)

Capability Development Division

Director General Maritime Development (Doc Data Sheet only)
Director General Land Development (Doc Data Sheet only)
Director General Aerospace Development (Doc Data Sheet only)

Army

ABCA Office, G-1-34, Russell Offices, Canberra (4 copies)
SO (Science), DJFHQ(L), MILPO Enoggera, Queensland 4051 (Doc Data Sheet only)

NAPOC QWG Engineer NBCD c/- DENGRS-A, HQ Engineer Centre Liverpool Military Area, NSW 2174 (Doc Data Sheet only)

Intelligence Program
DGSTA Defence Intelligence Organisation

Corporate Support Program (libraries)
OIC TRS, Defence Regional Library, Canberra
*US Defence Technical Information Center, 2 copies
*UK Defence Research Information Centre, 2 copies
*Canada Defence Scientific Information Service, 1 copy
*NZ Defence Information Centre, 1 copy
National Library of Australia, 1 copy

UNIVERSITIES AND COLLEGES

Australian Defence Force Academy
Library
Head of Aerospace and Mechanical Engineering
Deakin University, Serials Section (M list), Deakin University Library, Geelong
Senior Librarian, Hargrave Library, Monash University
Librarian, Flinders University

OTHER ORGANISATIONS

NASA (Canberra)
AGPS
State Library of South Australia
Parliamentary Library, South Australia

OUTSIDE AUSTRALIA

ABSTRACTING AND INFORMATION ORGANISATIONS
Library, Chemical Abstracts Reference Service
Engineering Societies Library, US
Materials Information, Cambridge Scientific Abstracts, US
Documents Librarian, The Center for Research Libraries, US

INFORMATION EXCHANGE AGREEMENT PARTNERS
Acquisitions Unit, Science Reference and Information Service, UK
Library - Exchange Desk, National Institute of Standards and Technology, US

SPARES (5 copies)

Total number of copies: **64**

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
DOCUMENT CONTROL DATA

1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)

2. TITLE

A Preliminary Study of Open Signalling for ATM Networks

3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION)

Document	(U)
Title	(U)
Abstract	(U)

4. AUTHOR(S)

T. A. Au

5. CORPORATE AUTHOR

Electronics and Surveillance Research Laboratory
PO Box 1500
Salisbury SA 5108 Australia

6a. DSTO NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DSTO-TR-0800	AR-010-882	Technical Report	March 1999

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. NO. OF PAGES	12. NO. OF REFERENCES
E8709/4/19 PT 2	ADF96/295	DGC3ID	22	14

13. DOWNGRADING/DELIMITING INSTRUCTIONS

14. RELEASE AUTHORITY

Chief, Communications Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE CENTRE, DIS NETWORK OFFICE, DEPT OF DEFENCE, CAMPBELL PARK OFFICES, CANBERRA ACT 2600

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CASUAL ANNOUNCEMENT Yes

18. DEFTEST DESCRIPTORS

Network control, Asynchronous transfer mode, Distributed computer systems, Signal theory (Telecommunication)

19. ABSTRACT

Based on a clear separation between switching hardware and control software, the concept of open signalling creates an open programmable networking environment. Network entities can be realised as high level objects with well-defined software interfaces, facilitating the creation of multiple mechanisms for connection management. Applying open signalling in defence networks can enhance the flexibility in supporting network services, allowing dynamic control of quality of service for maximum military value. This report discusses the design criteria and the associated performance issues of a connection management system for ATM networks. However, significant binding overheads are generated in remote operation invocations, adding a level of complexity to network control. This complexity is expected to diminish as middleware technology matures, thereby improving the overall performance in connection management.